Image Thresholding Binary Images have a lot of use cases in Image Processing. One of the most common use cases is that of creating masks. Image Masks allow us to process on specific parts of an image keeping the other parts intact. Image Thresholding is used to create Binary Images from grayscale images. You can use different thresholds to create different binary images from the same original image.

Function Syntax retval, dst = cv.threshold(src, thresh, maxval, type[, dst]) dst: The output array of the same size and type and the same number of channels as src.

The function has 4 required arguments:

src: input array (multiple-channel, 8-bit or 32-bit floating point).

thresh: threshold value.

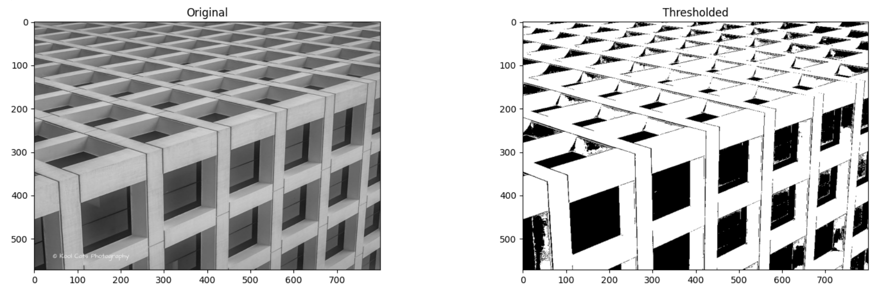
maxval: maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.

type: thresholding type (see ThresholdTypes).

Function Syntax

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img_read = cv.imread("Images/building-windows.jpg", cv.IMREAD_GRAYSCALE)
retval, img_thresh = cv.threshold(img_read, 100, 255, cv.THRESH_BINARY)

# Show the images
plt.figure(figsize=[18, 5])

plt.subplot(121);plt.imshow(img_read, cmap="gray");  plt.title("Original")
plt.subplot(122);plt.imshow(img_thresh, cmap="gray");plt.title("Thresholded")

print(img_thresh.shape)
```

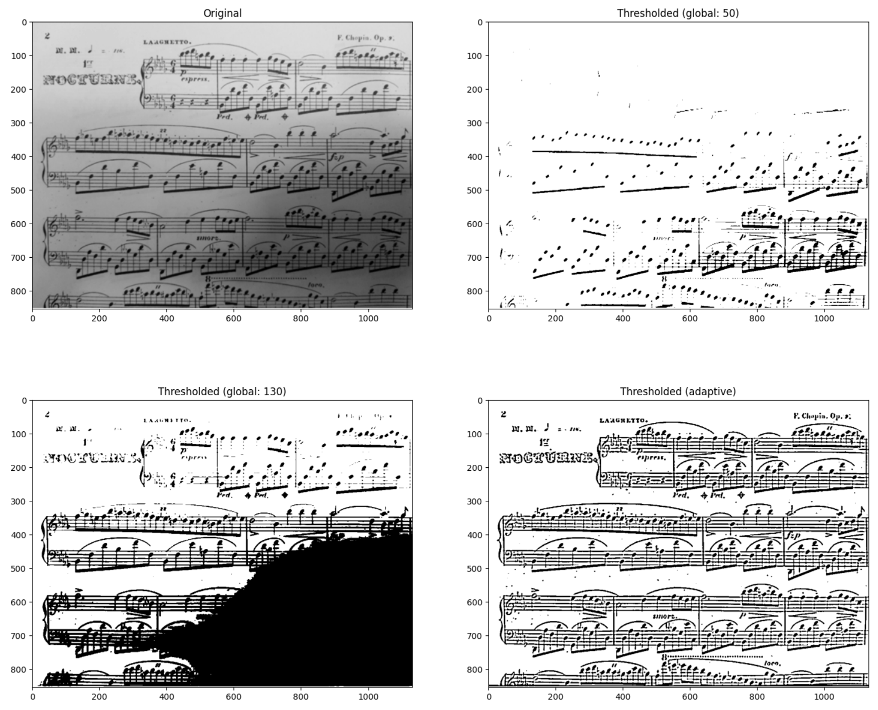
```
(572, 800)
```



Application: Sheet Music Reader

Suppose you wanted to build an application that could read (decode) sheet music. This is similar to Optical Character Recognigition (OCR) for text documents where the goal is to recognize text characters. In either application, one of the first steps in the processing pipeline is to isolate the important information in the image of a document (separating it from the background). This task can be accomplished with thresholding techniques. Let's take a look at an example.

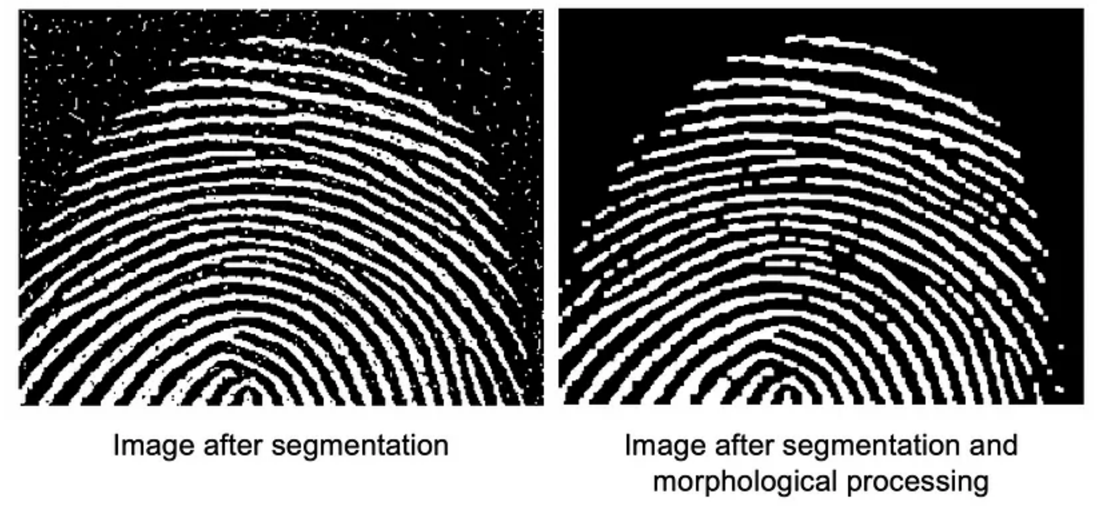
```
# Read the original image
img_read = cv.imread("Images/Piano_Sheet_Music.png", cv.IMREAD_GRAYSCALE)

# Perform global thresholding
retval, img_thresh_gbl_1 = cv.threshold(img_read, 50, 255, cv.THRESH_BINARY)

# Perform global thresholding
retval, img_thresh_gbl_2 = cv.threshold(img_read, 130, 255, cv.THRESH_BINARY)

# Perform adaptive thresholding
img_thresh_adp = cv.adaptiveThreshold(img_read, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 7)

# Show the images
plt.figure(figsize=[18,15])
plt.subplot(221); plt.imshow(img_read,        cmap="gray");  plt.title("Original");
plt.subplot(222); plt.imshow(img_thresh_gbl_1,cmap="gray");  plt.title("Thresholded (global: 50)");
plt.subplot(223); plt.imshow(img_thresh_gbl_2,cmap="gray");  plt.title("Thresholded (global: 130)");
plt.subplot(224); plt.imshow(img_thresh_adp,  cmap="gray");  plt.title("Thresholded (adaptive)");
```



Introduction to Morphology

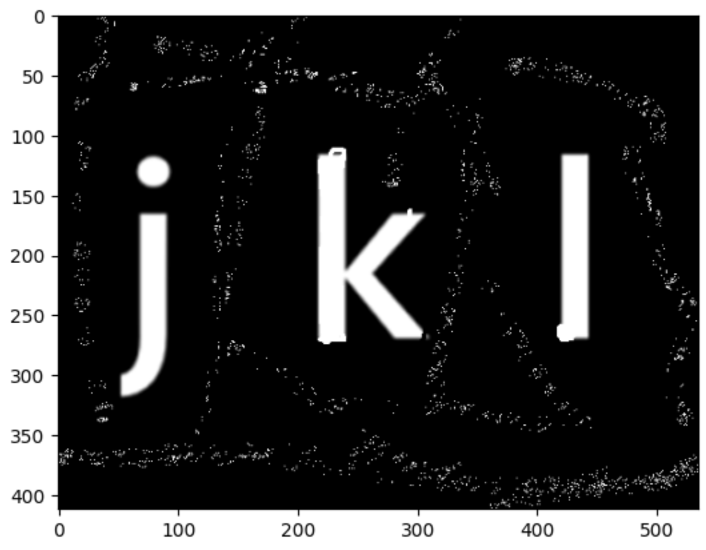
Morphology is a comprehensive set of image processing operations that process images based on shapes [1]. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors.

There is a slight overlap between Morphology and Image Segmentation. Morphology consists of methods that can be used to pre-process the input data of Image Segmentation or to post-process the output of the Image Segmentation stage. In other words, once the segmentation is complete, morphological operations can be used to remove imperfections in the segmented image and deliver information on the shape and structure of the image as shown in Figure 2.



```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('Images/three.jpeg', cv.IMREAD_GRAYSCALE)
img = 255-cv.imread('Images/jkl.png', cv.IMREAD_GRAYSCALE)

plt.imshow(img, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e2faf690>
```



Double-click (or enter) to edit

Double-click (or enter) to edit

1. Erosion

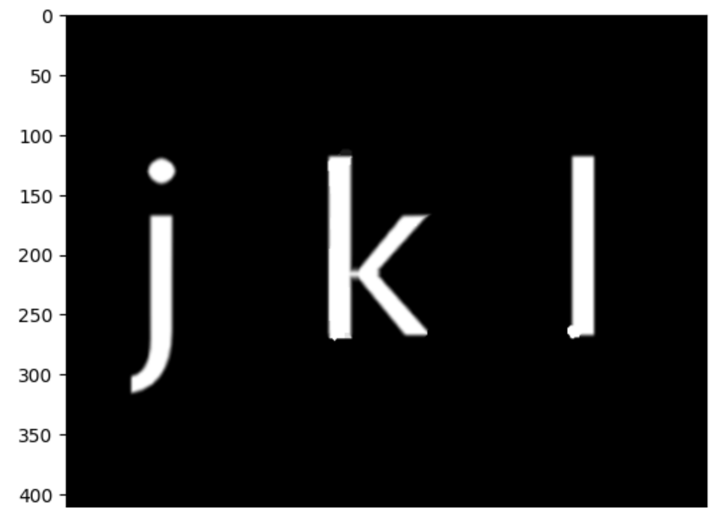
The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Here, as an example, I would use a 5x5 kernel with full of ones. Let's see it how it works:

```
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
plt.imshow(erosion, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e2354e90>
```



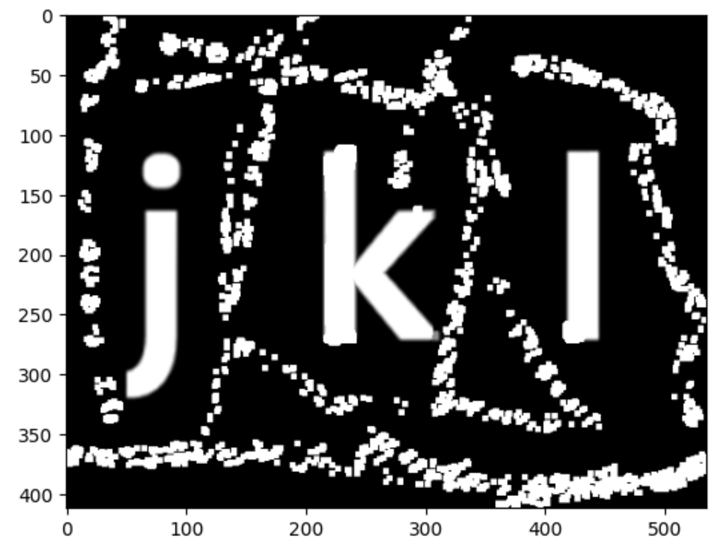
Double-click (or enter) to edit

2. Dilation

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

```
dilation = cv.dilate(img,kernel,iterations = 1)
plt.imshow(dilation, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e23cf090>
```



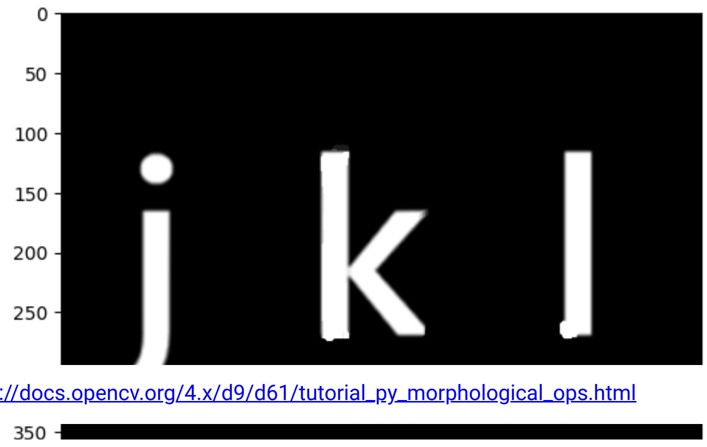
Double-click (or enter) to edit

3. Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above. Here we use the function, cv.morphologyEx()

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
plt.imshow(opening,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e2f54e90>
```



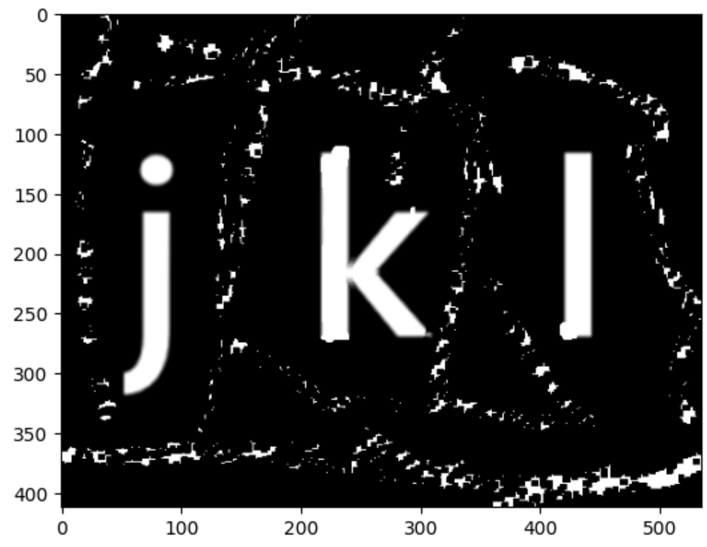
https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

4. Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object. closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)

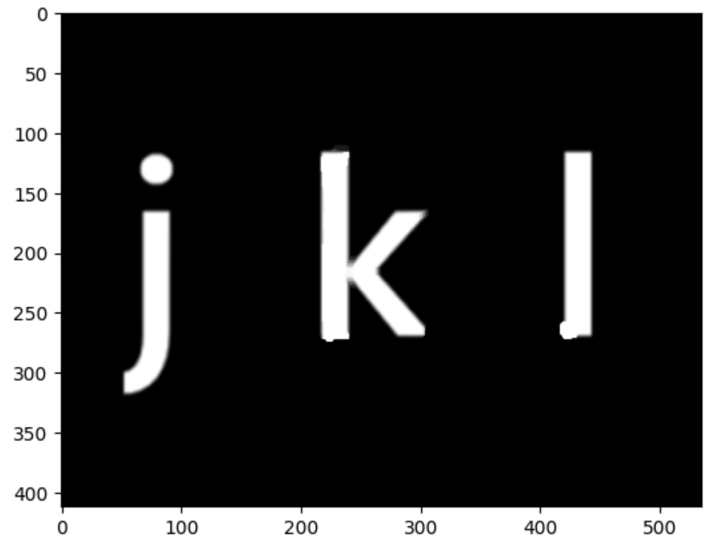
```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
plt.imshow(closing,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e22bf450>
```



```
closing = cv.morphologyEx(opening, cv.MORPH_CLOSE, kernel)
plt.imshow(closing,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e22a1410>
```



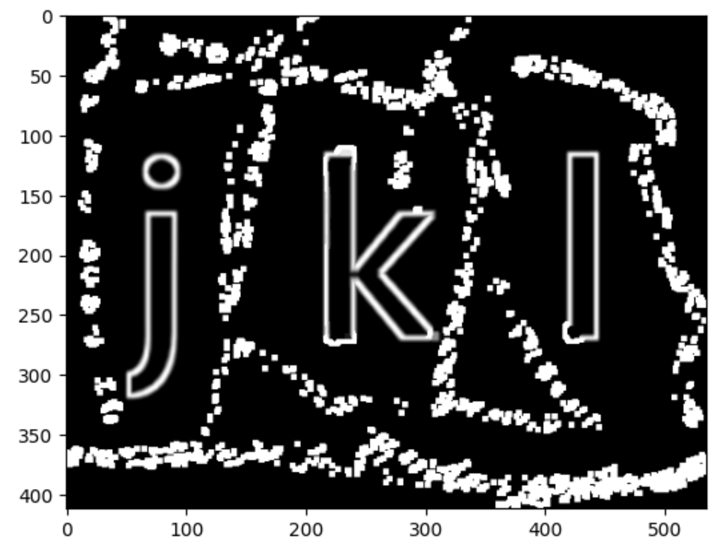
5. Morphological Gradient

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object. gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)

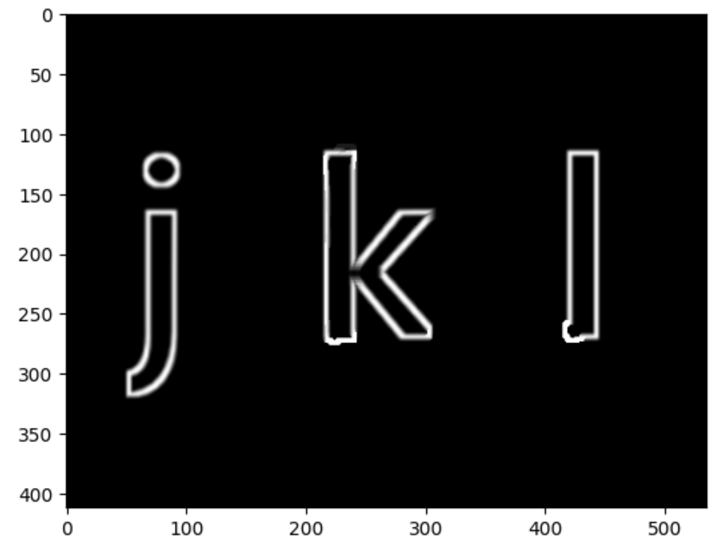
```
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)
plt.imshow(gradient,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132dfb81610>
```



```
gradient = cv.morphologyEx(closing, cv.MORPH_GRADIENT, kernel)
plt.imshow(gradient,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132dfb0b2d0>
```

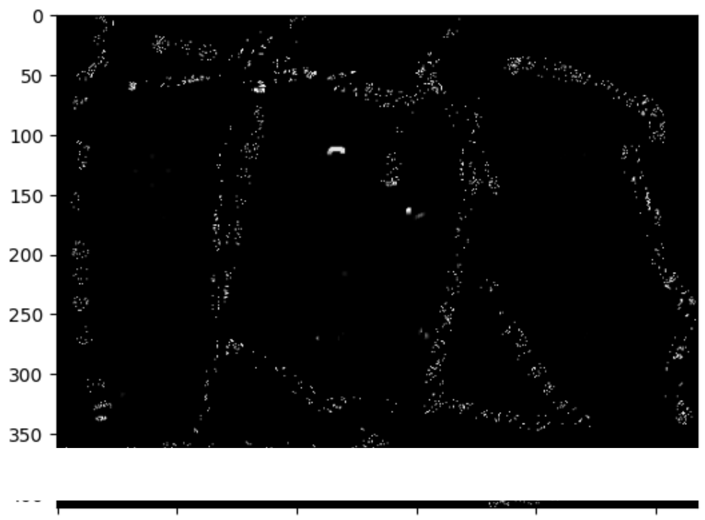


6. Top Hat

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel. tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)

```
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
plt.imshow(tophat,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e2225850>
```

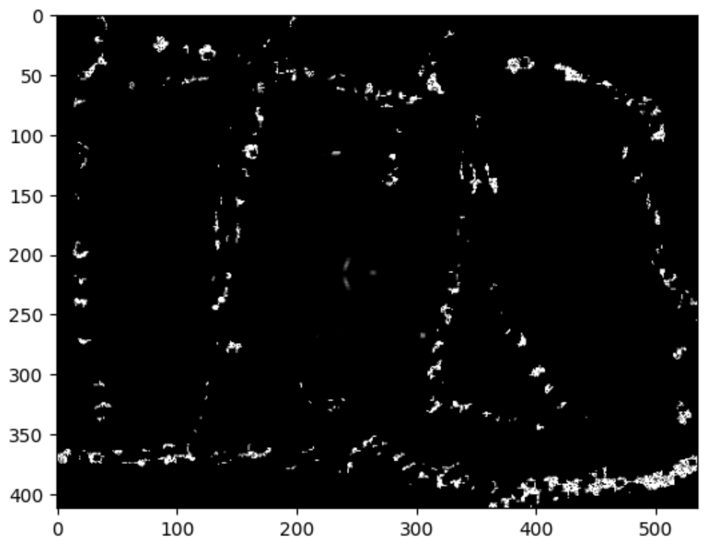


7. Black Hat

It is the difference between the closing of the input image and input image. blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)

```
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)
plt.imshow(blackhat,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x132e24b4fd0>
```



Structuring Element

We manually created a structuring elements in the previous examples with help of Numpy. It is rectangular shape. But in some cases, you may need elliptical/circular shaped kernels. So for this purpose, OpenCV has a function, cv.getStructuringElement(). You just pass the shape and size of the kernel, you get the desired kernel.

```
cv.getStructuringElement(cv.MORPH_RECT,(5,5))
```

```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)
```

```
cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
```

```
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

```
cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
```

```
    array([[0, 0, 1, 0, 0],
           [0, 0, 1, 0, 0],
           [1, 1, 1, 1, 1],
           [0, 0, 1, 0, 0],
           [0, 0, 1, 0, 0]], dtype=uint8)
```

https://homepages.inf.ed.ac.uk/rbf/HIPR2/hitmiss.htm

https://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm

https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm

https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm

https://towardsdatascience.com/understanding-morphological-image-processing-and-its-operations-7bcf1ed11756

https://towardsdatascience.com/understanding-morphological-image-processing-and-its-operations-7bcf1ed11756